Pacemaker: Fighting Selfishness in Availability-Aware Large-Scale Networks

Fabrice Le Fessant, Cigdem Sengul, Anne-Marie Kermarrec

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Pacemaker: Fighting Selfishness in Availability-Aware Large-Scale Networks

Fabrice Le Fessant — Cigdem Sengul — Anne-Marie Kermarrec

N° 6594 — version 2

initial version July 2008 — revised version January 2009

Thème COM

ISSN 0249-6399 ISRN INRIA/RR--6594--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
SACLAY - ÎLE-DE-FRANCE

Pacemaker: Fighting Selfishness in Availability-Aware Large-Scale Networks

Fabrice Le Fessant*, Cigdem Sengul*, Anne-Marie Kermarrec†

Thème COM — Systèmes communicants
Équipes-Projets Asap

Rapport de recherche n° 6594 — version 2 — initial version July 2008 — revised version January 2009 — 34 pages

Abstract: In this paper, we introduce Pacemaker, a scalable and lightweight protocol to measure reliably the availability of peers. To the best of our knowledge, Pacemaker is the only protocol resilient to the presence of selfish peers, i.e. peers lying about their availability and minimizing their contribution to the system. Pacemaker relies on a novel *pulse-based* architecture, where a small set of trusted peers regularly flood the network with pulses containing cryptographic values. Collecting these pulses enables peers to later prove their presence in the system at any time, using cryptographic signatures. This new architecture overcomes many limitations of ping-based systems, and can be easily deployed on ad-hoc networks and social-based topologies. Simulation results show that our protocol provides accurate availability measurements even in the presence of selfish peers. Furthermore, our results are verified by experiments in Planetlab, which also illustrate the deployability of Pacemaker in real networks.

Key-words: peer-to-peer, cryptography, availability, monitoring

* INRIA Saclay – Ile de France
† INRIA Rennes - Bretagne Atlantique

Centre de recherche INRIA Saclay – Île-de-France
Parc Orsay Université
4, rue Jacques Monod, 91893 ORSAY Cedex
Téléphone : +33 1 72 92 59 00

Pace-Maker: mesure de disponibilité d'un pair dans les réseaux large échelle

Résumé : La mesure de disponibilité dans un réseau pair-à-pair peut revêtir une très grande importance pour beaucoup d'applications collaboratives. Ainsi, cette information est inestimable pour identifier les pairs les plus stables, ou les groupes de pairs similaires par leur disponibilité. Cependant, comme de nombreuses applications veulent récompenser les pairs les plus stables, il existe une incitation claire pour les pairs à mentir sur leur disponibilité réelle. Dans ce papier, nous présentons un protocol léger et scalabe qui permet aux nœuds de mesurer la disponibilité d'un pair en présence de pairs égoïstes. Dans notre protocole, chaque pair est chargé de maintenir sa propre disponibilité en collectant des pulsations disséminées par une entité de confiance en utilisant des signatures cryptographiques. Celles-ci permettent à tout pair de vérifier par des challenges les informations de disponibilité transmise par un pair.

Mots-clés : pair-à-pair, cryptographie, disponibilité, monitorage

1 Introduction

A peer-to-peer network is composed of thousands of independent computers, which aggregate their resources over the Internet to run collaborative distributed applications. Such networks are subject to high dynamics: computers (peers) may join and leave arbitrarily or be subject to frequent disconnections. However, it has been observed that peers with high availability in the system are more likely to remain in the system for a longer time [4, 16, 23]. As a consequence, many peer-to-peer networks rely on peer availability to measure the stability of peers, and use this parameter to select peers for specific purposes. For example, the most stable peers can be elected as *super-peers* [9, 24], or as privileged peers to store replicas [5, 2, 7, 10].

Yet, to the best of our knowledge, current research does not address how availability can be measured securely and efficiently. As we discuss in Section 9, current systems either use expensive and incomplete measurement techniques, or rely on peers to give an honest estimation of their availability in the network. The fact that stable peers are usually rewarded in a peer-to-peer network creates a clear incentive to appear very stable, for instance by lying about the real availability, in order to be granted a better status in the network. Subsequently, such selfish nodes may get access to more resources than they should be able to access (i.e., free riders). For instance, in a peer-to-peer backup system, peers that lie about their stability might get undue and undeserved access to storage on the most stable peers in the system.

Motivated by these observations, we present a simple and lightweight protocol, called Pacemaker, to measure availability in a trusted way in peer-to-peer networks. The main idea of Pacemaker is to disseminate *pulses* by trusted peers in the network. These pulses are then used by peers as proofs of presence in the system at a given time. Essentially, through this simple scheme, peers are able to verify the accuracy of the availability claims by randomly challenging each other. Since challenged peers are expected to use the correct pulse for the queried period, Pacemaker is able to detect selfish peers trying to appear more available in the system. An overview of the protocol is given in Section 2, and the complete specification in Section 4.

In addition to providing high accuracy in availability measurements, Pacemaker is also highly scalable. It only requires that peers are connected to enough neighbors to form a redundant mesh to propagate the pulses. This requirement makes Pacemaker suitable for both structured (DHTs) and unstructured (gossip [24]) peer-to-peer networks, but also for topologies with limited communications, such as ad-hoc networks and social-based topologies [20]. Here, we focus on a simple mesh network, to illustrate that Pacemaker inherits the scalability characteristics of the underlying network. We describe our system model in further detail in Section 3.

The main contribution of our research is providing each peer a secure way to notify its availability to other peers in the system in a completely distributed manner and through local communications (i.e., communication is only necessary with neighbors) using standard cryptographic mechanisms. In this paper, we only consider the case of selfish nodes, which are trying to gain access to more resources than they are allowed to by appearing more available than they really are, for instance, by lying. For now, we did not consider the case where

nodes may collude to improve their availability, and we discuss this decision in Section 9.3.

We evaluated Pacemaker both through analysis, simulations and Planet-Lab deployment. Our simulation results, which are presented in Section 6, confirm that Pacemaker provides highly accurate availability information with very low cost for both real and synthetic workloads. Furthermore, Pacemaker remains highly scalable due to its light load. Section 7 presents the performance of Pacemaker under different kinds of selfish behaviors and shows that Pacemaker is still able to provide high accuracy. For instance, when 5000 peers out of 100,000 lie about their availability, Pacemaker is able to detect these nodes in less than 5 days by sending challenges only once a day and drive the error in availability measurements back to negligible. Similarly, Pacemaker is able to tolerate well the effect of 30% selfish peers, which stop disseminating pulses in the hopes of improving their availability by reducing the availability of their neighbors. Finally, we deployed Pacemaker on a 170-node Planet-Lab testbed, which again confirmed a very good match between the measured and the real availability (see Section 8). Based on our simulation results and our experience with Planet-Lab deployment, we conclude that Pacemaker provides a simple, low-cost, scalable and accurate way to measure peer availability in the presence of selfish peers.

2 Pacemaker in a nutshell

Pacemaker is a simple and lightweight protocol to track peer availability in a large-scale system. In Pacemaker, each peer is in charge of maintaining its own availability measure and providing it to other peers. Yet this declared availability can be arbitrarily checked in a peer-to-peer fashion in order to detect selfish peers.

In a nutshell, Pacemaker works as follows: a server is in charge of periodically disseminating *pulses* in the system, say one pulse per hour. Such pulses are propagated in the system once by all the peers in the network to their neighbors. Each peer maintains a list of the pulses it has heard of and uses this list to prove its availability in the system. Using this simple scheme, Pacemaker provides decentralized verification of peer availability in the presence of selfish peers.

Selfish behaviors considered in this paper include, for instance, trying to obstruct pulse dissemination or claiming, untruthfully, being connected to the system when not. To tolerate such selfish behaviors, pulses are generated and signed by a trusted entity. The signature certifies the association between the pulse and its diffusion time. Hence, when peers send their availability to other peers, they might receive a *challenge* in return. More specifically, a peer A may ask a peer B to provide a proof for a subset of the time periods that peer A claims it was available. A liar is detected easily since peers should be able to compute such a proof using the pulses corresponding to the challenged periods. Note that we do not consider the case where a peer propagates pulses indefinitely to provide other peers with pulses generated when they were not online. Such malicious behaviors are part of the problem of colluding peers, and just discussed in Section 9.3.

3 Model

3.1 Definitions

The goal of Pacemaker is to secure the measure of peer availability in a peer-to-peer network. *Peer availability* can be defined by two metrics according to the context:

- The ratio of time that the peer spent connected to the network, which is a value in the interval $[0, 1]$. This metric can be used directly to estimate the stability of the peer or its life expectancy.
- The intervals of time when the peer was connected to the network. This metric can be used to detect regular patterns in peer behaviors, predict future connections and disconnections, or differentiate between a temporary disconnection from a definitive departure.

Pacemaker provides an approximation of the second metric, from which the first one can be derived. Essentially, we define the *system availability* as the average availability over all peers in the network. Finally, a group of peers can be attached a specific *class of availability* depending on the associated application. For instance, peers with availability greater than 95% can be considered to be in the *super-peer* class.

In a system that favors highly available peers, peers may exhibit various selfish and malicious behaviors:

- **Opportunistic peers** only fulfill the steps of the protocol required to get a good status, but without impacting the status of other peers.
- **Lazy peers** only fulfill the steps of the protocol required to get a good status, and do so even when such a behavior can impact the status of other peers.
- **Lying peers** try to improve their own status by lying. They don't impact directly the status of others, but they might get undeserved access to resources.
- **Colluding peers** collaborate with each other either to improve their own status or to disrupt the system.

Pacemaker secures the measure of availability against the first three types of behavior, which are all selfish. Dealing with colluding peers is discussed in Section 9.3.

3.2 Network Model

We consider a large-scale network (more than tens of thousands of computers) composed of nodes (or peers), connected by a communication medium, typically IP. We assume there exists a logical overlay network where each node is aware of a small portion of the network, i.e. it knows the IP addresses of a set of D_{max} neighbors. This is typically the case in both structured and unstructured peer-to-peer networks. In the network, peers communicate by sending asynchronous messages. Although there is no bound on communication delays, most messages

are assumed to be received after a short delay and assumed to be lost after a longer delay. Although not a requirement, we expect nodes to connect through FIFO channels, which enforces sequentiality of messages. Additionally, there also exists a global clock, with which computer clocks are loosely coupled. This is necessary for a peer to know at which periods (the periods are considered system-wise) it was connected to the system. Hence, peers have an *approximate agreement* on time.

Pacemaker relies on the existence of a trusted entity. In this paper, we assume there exists particular peers, which are called the *servers*. Pacemaker does not require the non-server peers to know the identity of the servers or any other peer. However, it is assumed that the overlay network is connected enough to ensure that every peer in the network is reachable from at least one of the servers. Since servers have a specific role in the protocol, they may have a higher degree than D_{max}. This helps, for instance, to prevent Sybil attacks that try to circle servers to disrupt the diffusion of pulses. For the sake of simplicity, in the rest of the paper, we consider a single-server system. This assumption does not affect our results, since no communication is required between the servers.

3.3 Cryptographic Model

We assume that peers have access to strong cryptographic primitives, specifically for public-private key operations, which are the following:

- **generate_pair():** Generates a new pair of public-private keys. The common usage is that the private key, K_{priv}, is kept secret by the peer, while the public key, K_{pub}, is known to other peers in the system.
- **sign(`data`, $\mathbf{K}_{priv}$):** Returns a signature for `data` using the private key K_{priv}.
- **verify(S, `data`, $\mathbf{K}_{pub}$):** Verifies that S is a signature for `data` that was created using the private key K_{priv} associated with K_{pub}.
- **hash(`data`):** Returns the hash of `data`.

We assume that there is a special pair of keys, one public (called KS_{pub}) known by all peers in the system, one secret (called KS_{priv}) known only by the server. Each peer p in the system also owns a pair of keys, noted $K_{p,pub}$ and $K_{p,priv}$, to sign data. We also define $H_p = \mathbf{hash}(\mathbf{K}_{p,pub})$, and use it as unique identifier for p in the network. These keys should also be used by a peer-to-peer application running on top of Pacemaker to prevent selfish peers from easily changing their identity when they are detected. Furthermore, we assume there exists a way for peers to exchange their public keys by either using dedicated messages or due to cryptographic communication protocols already in place (such as TLS [6]). Finally, we assume these operations provide a high level of security (i.e., it is almost impossible to break the cryptographic properties of these functions by such as having a collision in the hash function) in the time limits needed for the application [15].

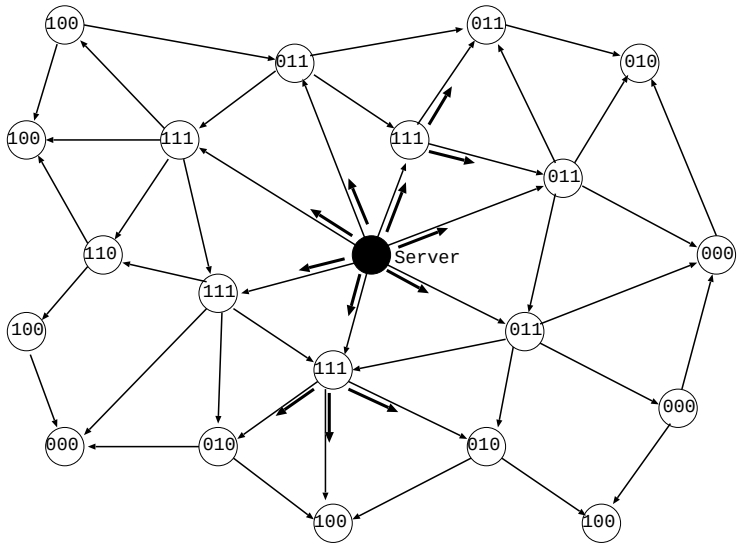

Figure 1: All peers are connected through a redundant mesh to the server. Here, the number in every node represents a 3-bit availability history as (3rd, 2nd, 1st) rounds. The arrows depict the pulse propagation. Every hour, a new pulse is propagated in the mesh by the server.

4 Pacemaker in Detail

Pacemaker is composed of three sub-protocols: (1) the pulse dissemination protocol; (2) the availability inquiry protocol and (3) the availability verification protocol, which are presented in their respective sections in the remainder of this section.

4.1 Pulse dissemination protocol

The server in Pacemaker is in charge of generating one *pulse* over a given period of time P. The dissemination of a pulse consists of each peer forwarding it once to all its neighbors. An example of the pulse dissemination is depicted in Figure 1. The figure shows a redundant mesh network, where there exists multiple paths between each peer in the network. This redundancy is essential to decrease the impact of peers that do not follow the protocol (i.e., the impact of nodes that do not forward the pulse to their neighbors).

A pulse T^i, which is generated by the server for time i, is a tuple (i, K^i_{pub}, K^i_{priv}, S^i), where K^i_{pub} and K^i_{priv} is a new fresh public-private key pair. The public-private key pairs are generated by the server on-demand. The pulse also includes S^i, which is a signature of (i, K^i_{pub}) using the servers' private key KS_{priv}. The server diffuses the pulse to its neighbor set (NS) at time i (code Fig. 2).

Every peer keeps an history of these pulses, representing its presence in the network during a window of time $\mathrm{N}_t \times P$. On receipt of a new pulse, a peer first checks if it has already received the same pulse and if not, verifies the

Server at time i:

```
let (K^i_pub, K^i_priv) = generate_pair();
let S^i = sign( <i, K^i_pub>, KS_priv );
let T^i = Pulse(i, K^i_pub, K^i_priv, S^i);
∀q ∈ NS_server, send( q, T^i );
```

Figure 2: Pulse generation at the server.

Node p receiving $\mathbf{T}^i$= Pulse(i, $\mathbf{K}^i_{pub}$, $\mathbf{K}^i_{priv}$, $\mathbf{S}^i$):

```
if T^i ∉ History_p and
     verify(S^i, <i, K^i_pub>, KS_pub )
then
  add(History_p, T^i);
  ∀q ∈ NS_p, send( q, T^i );
end if
```

Figure 3: Pulse diffusion by a peer.

Node p sending to q its availability at time i:

```
let bits = new bitfield[N_t];
for x in [1..N_t]
  if ∃ T^j ∈History_p | j ∈ [i − xP, i − xP + P[ then
    bits[x] := 0
  else
    bits[x] := 1
  end if
end for
let S = sign( < i, bits >, K_p,priv )
send( q, Availability(i,bits,S) );
```

Figure 4: Advertisement of availability by a peer.

authenticity of the pulse. If the pulse is indeed generated by the server, it updates its history and forwards the pulse to its neighbors (code Fig. 3).

4.2 Inquiry Protocol

Depending on the application, peers need to be able to check the availability of other peers. This might be done either regularly or just the first time they connect to each other. The verification of availability requires knowing the pulse history of peers. For this purpose, each peer sends a message `Availability`(i,bitfield,S), where i is the current time, and bitfield is an array of bits of size N_t, containing, for each period, 1 if it has the pulse, and 0 otherwise (code Fig. 4). The message also contains a signature of the bit field using the peer private key, $\mathrm{K}_{p,priv}$. This signature can be used to prove later that the message was sent by the peer, in particular, if the peer does not reply to a challenge.

Node p receiving Challenge(i, nonce) from node q:

```
if Pulse(i, K^i_pub, K^i_priv, S^i) ∈ History_p then
  let reply = sign( < nonce, H_p, H_q >, K^i_priv )
  send( q, Proof(i, nonce, K^i_pub, S^i, reply) );
end if
```

Figure 5: Reply to a challenge by a peer (one bit to simplify)

Node q receiving Proof(i,nonce,$\mathbf{K}^i_{pub}$,$\mathbf{S}^i$, reply) from node p:

```
if (i, nonce, p) ∈ challenges_q and
     verify(S^i, <i, K^i_pub>, KS_pub ) and
       verify(reply, < nonce, H_p, H_q >, K^i_pub )
then
   good_reply(p)
else
   bad_reply(p)
end if
```

Figure 6: Verification of a proof by a peer (one bit to simplify).

Using the bit field of another peer, a peer can compute an approximation of the availability of the peer during the period (N_t * P) by counting how many bits are set to one. Note that, in fact, the bit field only proves that the peer was online when the pulses were propagated and not during complete hours. However, we show in our simulations that sending the pulse at a random time in the current period provides a very good approximation of the real availability.

4.3 Verification Protocol

It is in the interest of some peers to lie about their uptime, especially to get more resources than they deserve. We thus provide a verification scheme to allow a peer to verify that the bit field received from another peer is correct. More specifically, to challenge a given peer, a peer selects one bit set to 1 in the bit field received from this peer (this can be easily generalized to several bits challenged at once). It sends a special request `Challenge`(i,`nonce`), containing i, the period of the bit to be verified, and `nonce`, which is a randomly generated short string to make the challenge unique. On reception of `Challenge`(i,`nonce`), the peer replies `Proof`(i,`nonce`, K^i_{pub}, S^i,`reply`) where K^i_{pub} and S^i are respectively the public key and the signature from pulse T^i, and `reply` is the signature of `nonce` and the hashes of the identities of the two peers by the the private key K^i_{priv} (code Fig. 5). On reception of `Proof`(i, `nonce`, K^i_{pub}, S^i, reply), the peer can verify that `reply` is the signature with the correct key K^i_{priv}, using the key K^i_{pub} from the message, and can also verify that K^i_{pub} is the public key from the pulse T^i using the signature S^i and the well known KS_{pub} key (code Fig. 6).

The actions to be taken when a peer fails to provide a correct reply to a `Challenge` message is out of the scope of this paper since it mostly depends

Trace	Size	Length	Sessions			Availability			Absolute Error		
	peers	days	< 1h	—	> 1d	$< 25\%$	—	$> 75\%$	$< 1\%$	—	$> 3\%$
Skype Superpeers[11]	2081	29	15%	80%	5%	60%	22%	18 %	95%	4%	1%
Microsoft Desktops[3]	51663	10	0%	85%	15%	15%	15%	70%	97%	3%	0%
Overnet Clients[1]	1469	7	39%	61%	0%	49%	44%	7%	58%	32%	10%
Synth. Uniform	100000	20	30%	68%	2%	20%	60%	20%	20%	70%	10%
Synth. Exponential	100000	20	27%	61%	2%	80%	15%	5%	20%	78%	2%

Figure 7: We ran simulations using a few availability traces collected for different workloads. Except the Skype workload, such systems are not representative of real applications for Pacemaker: Microsoft network exhibits a very small churn, typical of company networks, whereas on the contrary, Overnet's churn is very high, even for a file-sharing application (actually much higher than recent observations on the Edonkey network).

on the application using our measurement system. However, our protocol is designed so that it is possible to propagate both the `Availability` and the `Challenge` messages to other peers in the network. Hence, other peers are allowed to use a not replied `Challenge` message to challenge the same peer again. To avoid false claims, the peers might use the `Availability` messages to check that the message was indeed signed by a selfish peer. If some pulses are damaged on a peer due to a failure, thus preventing verification, the peer is expected to clear the corresponding bits from its availability history.

5 Evaluation Road-map

The main goal of our evaluation study is to illustrate that Pacemaker is:

- **Scalable**: It can accommodate the growth of the system; it is able to work with millions of peers connected together.
- **Accurate**: The error between the measured and real availability of a peer is negligible.
- **Low-cost**: It is less expensive than other systems providing a similar measure.
- **Secure**: The measure still reflects the reality even though selfish peers may try to modify it. Furthermore, it is able to detect lying nodes timely.
- **Easy-to-deploy**: It can be implemented easily and deployed with minimal configuration on the peers.

We studied Pacemaker via a combination of analysis, simulations using real and synthetic traces and an implementation on a 170 node Planetlab testbed. In the remainder of this paper, we first present the performance of Pacemaker as an availability measurement system using simulation results on the synthetic traces. Although, we also ran simulations with real traces (see Fig. 7), we omit these results for the sake of brevity, especially because synthetic traces allow us to evaluate Pacemaker on larger-scale networks (100,000 peers) with more extreme availability distributions (uniform and exponential).

Next, we present results where Pacemaker operates in the presence of selfish peers that try to cheat the system by advertising a higher availability. Our goal in these experiments is to show that Pacemaker is able to provide accurate availability measurements in an efficient manner even in the presence of such selfish peers. Finally, we conclude with a discussion on implementation details. Essentially, the Planetlab experiments illustrate the ease of deployment of Pacemaker in a realistic setting.

6 Availability Monitoring with Pacemaker

The first goal of our evaluation study is to prove the scalability of Pacemaker, the accuracy of its availability measurements and the negligible load it adds to the system. To this end, our simulation setup consists of two parts: (1) the availability patterns of peers and (2) the unstructured overlay network (the mesh) connecting peers and the server. In this section, we first present this setup in detail and next, the performance results in comparison to a ping-based availability measurement system.

6.1 Simulating the Availability of Peers

In the synthetic traces used by our simulations, availability follows either a uniform or exponential distribution. While working with the uniform distribution allows us to span all values of availability, the exponential distribution is more representative of real peer-to-peer systems [1]. Based on these two distributions, the availability of a peer y, a_y, is calculated as:

$$a_y = \begin{cases} 0.02 + 0.98 \cdot U(0,1) & \text{if uni.} \\ \max(0.02, \min(1, e^{1-\ln(2+65 \cdot U(0,1))})) & \text{if exp.} \end{cases} \quad (1)$$

Additionally, the number of disconnections per day, d_y for each peer follows a uniform distribution: $d_y = U(0, 10)$. Using a_y and d_y, the probabilities to switch between "ON" (i.e., online and available) and "OFF" (i.e., offline and not available) states are computed as follows.

$$\lambda_y = \frac{d_y}{24 \times 60} \quad (2)$$

$$\mu_y = \frac{a_y \times \lambda_y}{1 - a_y} \quad (3)$$

Using these two probabilities, the Markov chain depicted in Fig. 8 drives the state changes. Additionally, to account for the effect of the timezones, this Markov chain is modified to obtain a diurnal pattern: during the day, peers have twice their normal probability of switching to ON and half their normal probability of switching to OFF.

The resulting availability patterns are presented in Fig. 9 and Fig. 10, which depict the number of online peers and the session lengths of peers, respectively, for uniform and exponential distributions. Fig. 9 shows that the number of available peers per round is lower with exponential distribution compared to uniform distribution. Essentially, while with exponential distribution, the number of available peers per round is approximately 25,000 during the day and 10,000 during the night, for the uniform distribution, the number of available

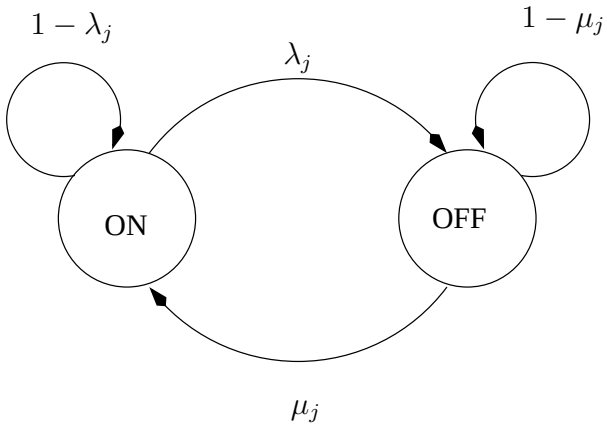

Figure 8: Online and offline times of a peer are computed using a Markov chain with probabilities λ and μ.

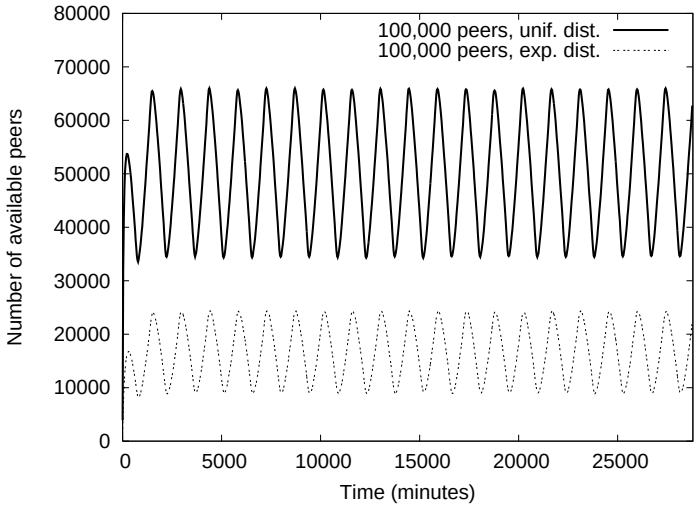

Figure 9: Number of peers online over time. Since the timezones of peers are only on 12 hours, the number of peers follow a diurnal pattern, which would not be the case if the distribution was over 24 hours.

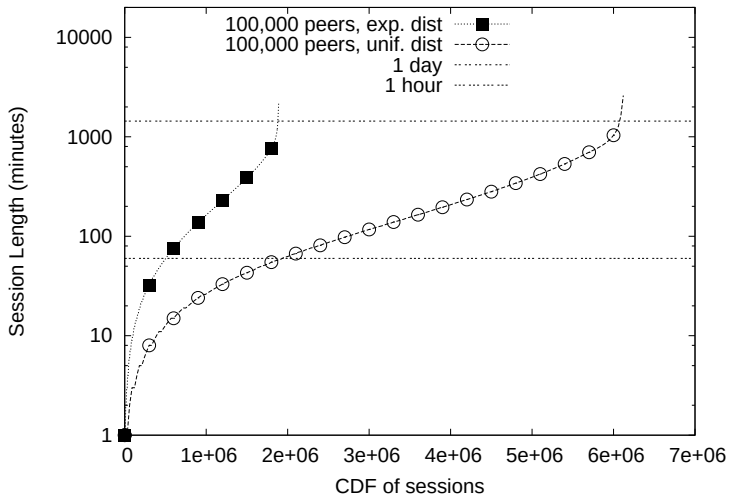

Figure 10: CDF of Sessions lengths. The median session length is around two hours.

peers is 65,000 during the day and 35,000 during the night. Fig. 10 shows that, as expected from Fig. 9, there are a higher number of sessions in the case of uniform distribution compared to exponential distribution. The session lengths range from a minute to a few days for both distributions and the median session length is around two hours.

6.2 Overlay Network Setup

We use a mesh that connects all peers and a single server. The server diffuses pulses through this mesh every hour (i.e., unit time P = 1 hour) to measure the availability of peers. In our simulations, the mesh is formed as follows: the server has an out-degree of 10 (called children in the sequel), and each peer has a out-degree (children) and in-degree (parents) of 5. Although many other approaches could be considered to build the mesh, we used the simple following protocol: to connect to the mesh, a peer first sends an `AskRoot` message to the server, which replies with a list of its children in the graph. The peer then sends `AskParent` messages to the children. Every child either accepts the peer as a child, or sends a random child among its children. The process iterates until the peer is connected to 5 different parents.

We added the following local optimizations to improve the mesh:

- At every round, if a peer has a free child slot, it chooses among all the child candidates the one with the best measured availability. To this end, in our simulations, each peer delays its response to `AskParent` messages by 1 minute to be able to choose the best candidate.
- To decrease the diameter of the network, a peer disconnects the children that are at the same distance from the server. The distance information is learned either from the `AskParent` or the `Distance` update messages, which are sent by a peer each time its distance to the server changes.

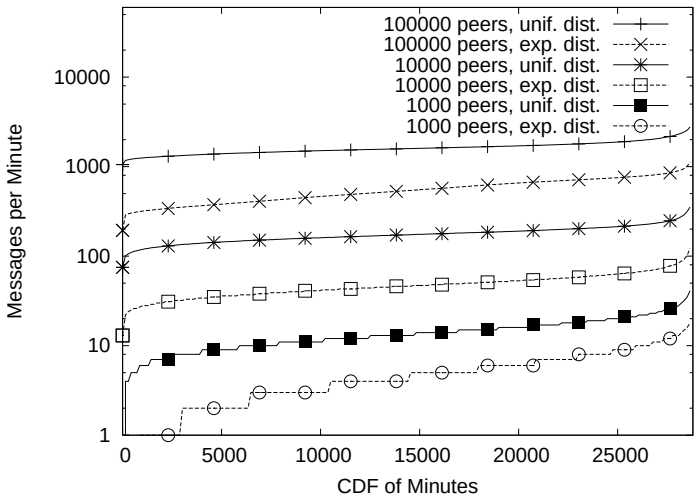

Figure 11: The number of `AskRoot` messages received by the server per minute in our simulations. The mean rate is 1/50 of the number of nodes, i.e. 100,000 nodes consume 33 messages per second. Even in our simplified mesh, a few servers can easily handle a few millions of peers.

> Obviously, for peers that do not have a parent, the distance to the server is infinity. Otherwise, peers advertise their minimum distance to the server through their current parents. Note that the well-known *count-to-infinity* problem might occur during the `Distance` updates and is resolved similar to [12] by choosing a small number (e.g., 10) for *infinity*.

While we chose this specific mesh generation protocol here for its simplicity, the mesh could be built differently based on the application requirements and the service desired. Pacemaker only requires the underlying mesh to be able to diffuse the pulses successfully, which is a reasonable expectation. Since Pacemaker relies on diffusion of pulses, the scalability and efficiency of our protocol and the accuracy of the availability measurement depends on the underlying mesh. Therefore, we first show the scalability of the mesh used in our simulations by counting the number of `AskRoot` messages received by the server from new or reconnecting peers (see Fig. 11). We evaluate the number of `AskRoot` messages for both uniform and exponential availability distributions with 1,000, 10,000 and 100,000 peers. As expected, as the number of peers increases, the number of `AskRoot` messages also increases. Furthermore, with uniform distribution, since the number of sessions is higher, we observe a higher number of messages sent per minute. This is because the peers in our simulations are memoryless and hence, each time they come back online, they need to rediscover parents. Nevertheless, even with this property, Fig. 11 shows that the mesh is scalable: the number of `AskRoot` messages grows to only 1000 when the number of peers increases to 100,000. Note that this basically translates to less than 10 kB/s traffic load on the server, which is very reasonable.

To understand the scalability of the constructed mesh further, Fig. 12 plots the maximal hop-distance to the server. As expected, it grows logarithmically

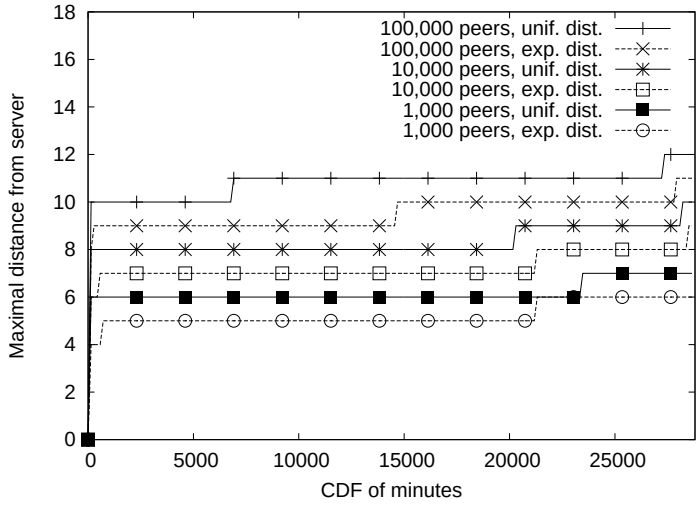

Figure 12: Maximal distance to the server over time. Note that the diameter of the network is not too high.

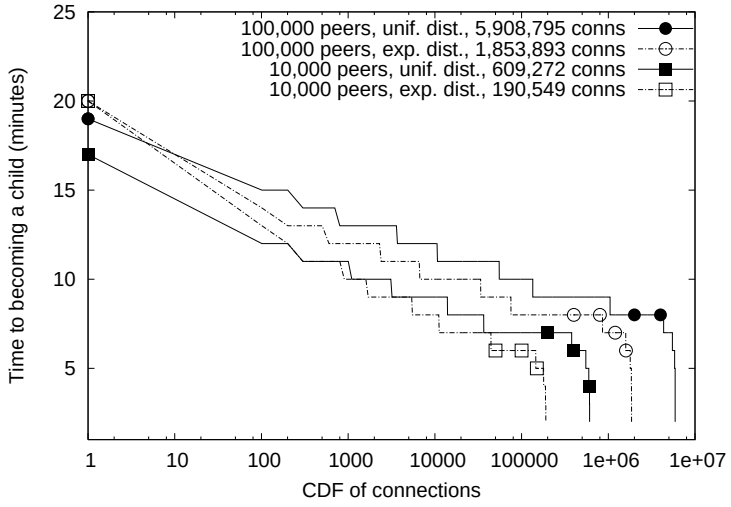

Figure 13: Time spent between the beginning of a peer session and the connection to its first parent. Only in some rare cases, it is above 7 minutes. Since in our system, we focus on long session times (median session length is two hours), this delay is negligible.

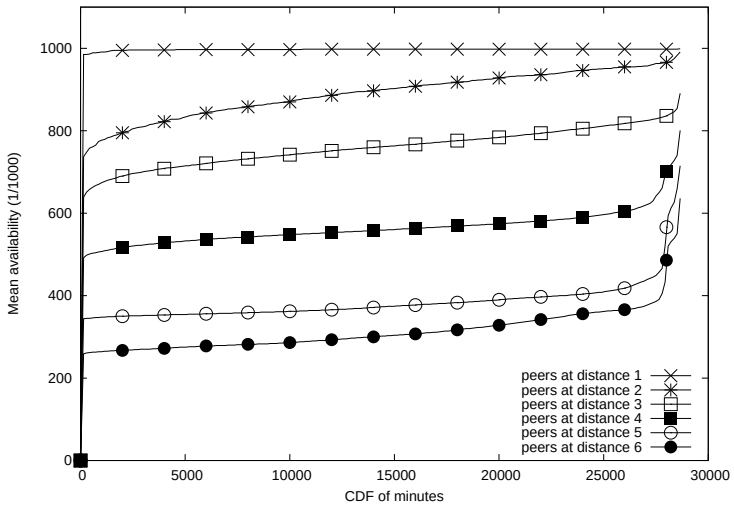

Figure 14: Mean availability of connected peers depending on their distance to the server. The peers closer to the server have a higher availability that farther peers.

with the number of peers in the network. More specifically, the maximum number of hops range between 5 and 12 depending on the number of peers and the availability distribution. The effect of this is also seen in Fig. 13, which shows the time it takes for a new peer to find its first parent. This delay is more than 10 minutes only for less than 1/1000 of the connections. Essentially, these delays, which are on the order of a few minutes, can be considered as negligible, since we are interested in measuring availability for long sessions (i.e., the median session length is two hours).

Finally, the effect of our local optimizations are depicted in Fig. 14, where the mean availability of peers versus their distance to the server is plotted. The figure shows that peers close to the server have a higher availability than peers farther away. This is due to prioritizing peers with higher availability when selecting children. Without such a prioritization, we would not observe this effect and the mesh would be less stable.

6.3 Results on Accuracy

In this section, we present results in terms of accuracy and cost-effectiveness of Pacemaker. We simulate 100,000 peers for 20 days (i.e., 28,800 minutes). Every round in the simulation takes one minute. From a communication point of view, this puts some timeout on messages, which allows us to detect peer disconnections (for instance, TCP keepalive is 30 seconds).

For each peer in the network, we compute the accuracy as the difference between its real availability and the availability measured by our system (i.e. the availability that it is able to prove to other peers). Essentially, this represents the absolute error, plotted on Fig. 15 for both the uniform and exponential distributions. Our measured availability matches the real availability of peers when the pulse period, P, is 1 hour. Uniform random distribution exhibits the

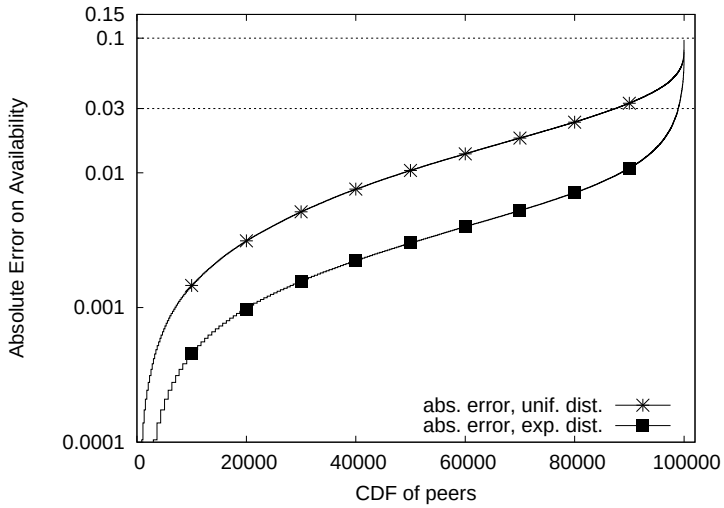

Figure 15: Measured availability compared to real availability for uniform distribution. Note that the difference is negligible.

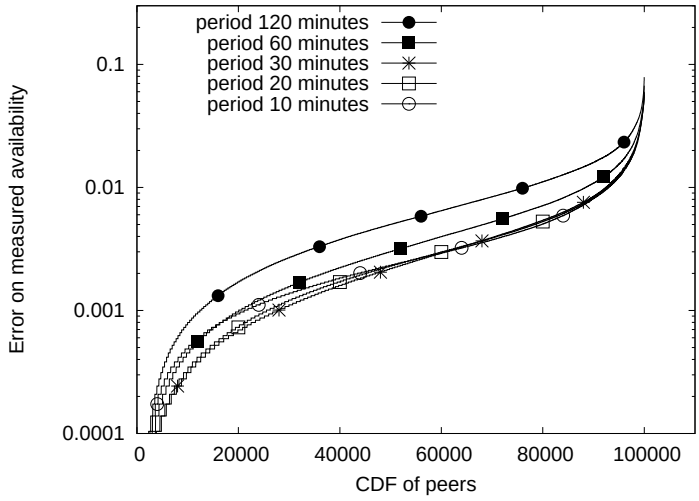

Figure 16: The error in measured availability versus P for exponential distribution

worst case: for 70% of the peers, the absolute error is less than 1%, for the next 20% of the peers, it is less than 3%, and never exceeds 10% (of time).

Obviously, if we reduce the pulse period P, we can achieve better accuracy but at a higher cost. Fig. 16 and Fig. 17 show the accuracy of Pacemaker as P ranges between 10 minutes to 2 hours. For both distributions, while P = 2 hours would achieve the best cost, it also provides the lowest accuracy (i.e., the error is significantly higher). While the error immediately improves with P = 1 hour, for lower P values, Pacemaker performs with comparable accuracy. This

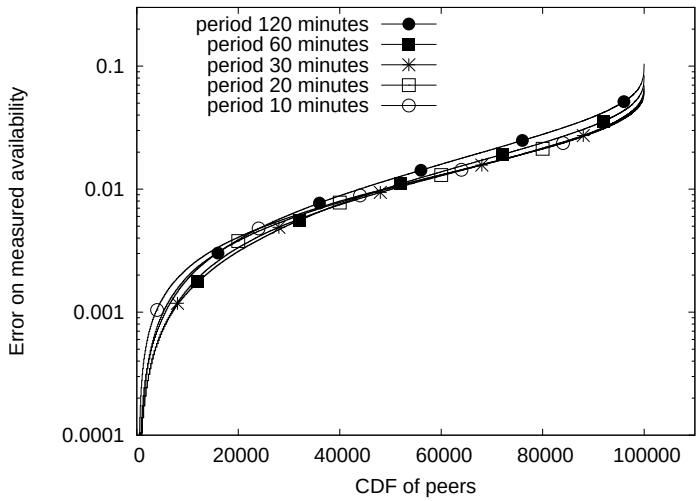

Figure 17: The error in measured availability versus P for uniform distribution

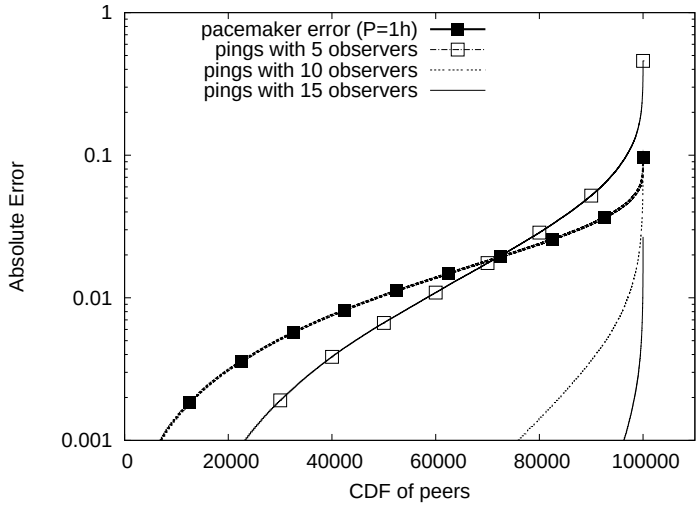

Figure 18: A comparison between Pacemaker and ping-based system for uniform distribution. Pacemaker is equivalent to a ping-based system that uses 5 and 10 observers for each peer.

shows that Pacemaker achieves a good trade-off between accuracy and cost. Note that in our simulations, peers accept unordered pulses (i.e., a pulse for a given time would be accepted even if a pulse for a later time has already been received). Not following this policy can degrade the accuracy of the measure for small values of P as the peer distance to the server increases.

We believe that even in the worst case, the accuracy of Pacemaker is acceptable for a majority of the applications since 1-5% error does not change the

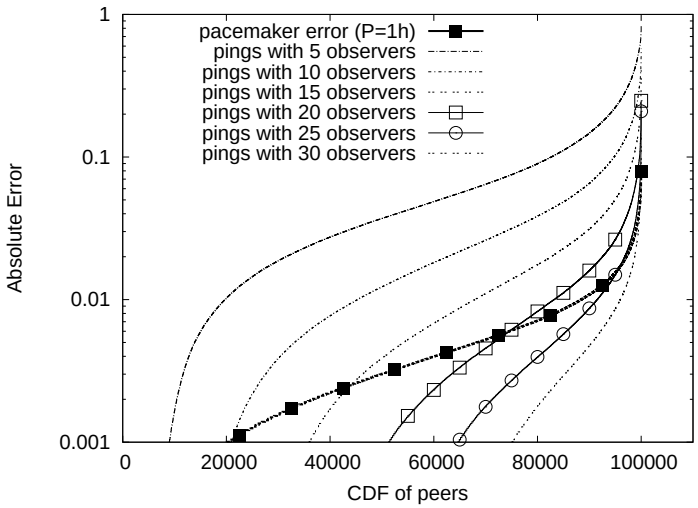

Figure 19: A comparison between Pacemaker and the ping-based system for exponential distribution. Pacemaker performs better than a ping-based system with 20 observers for each peer.

availability *class* of a peer (as discussed in Section 3). Furthermore, we compare Pacemaker with a system where each peer in the network is monitored every minute using pings by a small set of randomly selected *observers* (similar to AVMON [19]). Fig. 18 shows that for the uniform distribution, the ping-based system achieves high accuracy (i.e., negligible error) with only a low number (5-10) observers. However, this does not hold for the more realistic exponential distribution, as seen in Fig. 19. In this case, the ping-based system must use a higher number of observers (e.g., 20 observers) to reach the same accuracy as Pacemaker. On the other hand, in the worst case, Pacemaker has a cost of 10 `Pulse` messages per peer (if the mesh degree is 10) and per hour (if the period P is one hour). Furthermore, note that in a ping-based system, no measurements can be taken if the observers are down, which is often the case for the exponential distribution. Therefore, Pacemaker provides more accuracy as its availability measurement does not depend on a fixed set of observers.

7 Pacemaker against Selfish Peers

While Pacemaker achieves good accuracy in environments where no selfish peers are present, it is essential to maintain similar performance when peers exhibit selfish behavior. In the following section, we evaluate how Pacemaker handles different selfish behaviors, which are identified in Section 3. These behaviors, translated into Pacemaker context, are namely:

- **Lazy peers:** These peers do not propagate pulses so that other peers have a lower measured availability.
- **Opportunistic peers:** These peers only connect to the mesh to receive the pulses and immediately disconnect afterward.

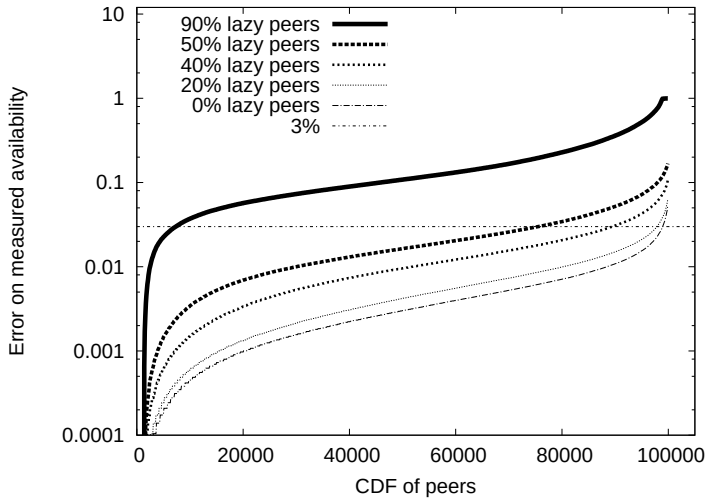

Figure 20: Error on measured availability depending on the number of lazy peers in the system. Although lazy peers do not propagate the pulses to their neighbors, the error is under 10% with 50% of lazy peers in the network, and almost not affected with 30% of lazy peers.

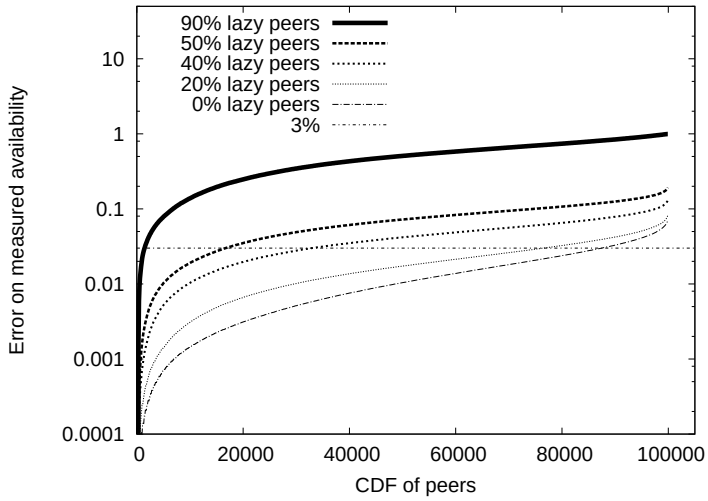

Figure 21: Error on measured availability with lazy peers with a uniform distribution

- **Lying peers:** These peers lie about their availability.

7.1 Lazy Peers

Lazy peers prevent other peers from benefiting from the system by not propagating the pulses to their neighbors. Basically, lazy peers do not follow the protocol when the protocol actions have a cost and no direct benefit. As an

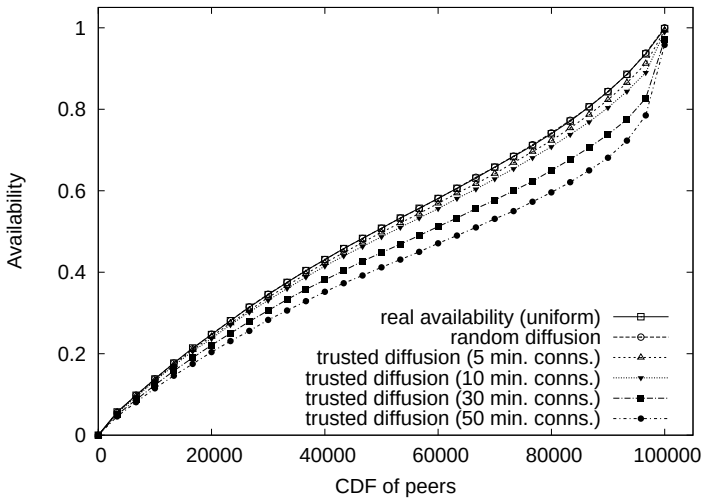

Figure 22: Availabilities measured using either random checks or by requiring some connection length. It shows that sending the pulse randomly (at a random minute during the pulse hour) performs much better than requiring a minimal session length (from 5 to 50 minutes) from a child to propagate the pulse.

additional consequence, these peers may improve their ratings in the system by decreasing the measured availability of other peers rather than claiming a higher availability like the lying peers. To evaluate the impact of lazy peers, we randomly selected peers as lazy peers in our simulations. The results, depicted in Fig. 20, show that the accuracy of the availability measure is not affected much (i.e., the error remains under 10%) until the percentage of lazy peers hits 50%. This good results are a direct property of the degree chosen for our simulation mesh. Hence, we conclude that:

- When there is a low number of lazy peers (i.e., up to 30% of lazy peers), their impact is negligible on the measured availability of other peers. Hence, lazy peers do not succeed in reducing the availability of their neighbors.
- When there is a high number of lazy peers (i.e., greater than 50%), their impact is more significant but their measured availability is also as diminished as the one of collaborative peers since they are also affected by other lazy peers that do not propagate pulses.

7.2 Opportunistic Peers

Opportunistic peers try to cheat the system by connecting to the network only to get pulses to increase their perceived availability. In our system, such peers would connect at fixed times, depending on the schedule of pulse diffusion. To avoid opportunistic behavior, we propose two different policies:

- **Random diffusion:** Within each period P, the server starts the pulse diffusion at a random time. Hence, the opportunistic peers cannot forecast when to connect to the network.
- **Trusted diffusion:** When a peer receives a pulse, it only propagates the pulse to children which have been connected for a long time. Hence, opportunistic peers never receive pulses.

Fig. 22 plots the impact of these policies on the accuracy of measured availability. It shows that random diffusion performs much better as soon as the required session length for children becomes too long. Essentially, trusted diffusion requires guessing the average neighbor session length to avoid punishing good neighbors that do not have long session length. This might be difficult since the session lengths exhibit a high variance. Hence, we used random diffusion in all our simulations.

7.3 Lying Peers

In contrast to lazy and opportunist peers, *lying peers* try to achieve a higher status in the network by advertising false availability information. In our system, this is simply done by switching a 0 bit to 1 in the availability bit field. Pacemaker provides peers with the ability to challenge peers based on their advertised availability. Using this scheme, the probability that a challenger y discovers that x is lying in a given try is determined by two factors:

- How many bits x lied about (i.e., how many bits are switched from 0 to 1)
- How many bits y challenges

In this paper, we did not consider the case of challenging the entire bit field due to the high computational overhead and the growth in message size. Hence, in a given try, only a fixed number of bits, denoted as i, are challenged. Our goal is to calculate the probability that a challenge sent to peer x succeeds when i bits are challenged. Given the number of switched bits, $n_{switched}$, and the number of correct bits, $n_{correct}$, (which add up to the total number of 1-bits in the bit field) $p(x, 1)$ can be calculated as:

$$p(x,1) = \frac{n_{switched}(x)}{n_{correct}(x) + n_{switched}(x)} \tag{4}$$

This can be generalized to i bits as follows. A challenge would not succeed if and only if all the challenged i bits are correct. Hence,

$$p(x,i) = 1 - \Pi_{k=0}^{i-1} \frac{n_{correct}(x) - k}{n_{correct}(x) + n_{switched}(x) - k} \tag{5}$$

Fig. 23 shows how $p(x, i)$ increases with i when the percentage of $n_{switched}$ bits among the total number of 1-bits is 5, 10, 25 and 50%, respectively. Note that when the lying percentage is low, $p(x, i)$ is also low - even though it improves with increasing i. On the other hand, at a given challenge, when the lying percentage is high, it is more probable to detect liars even when i is low. For instance, for $i = 3$, $p(x, i) = 0.89$ when lying percentage is 50%.

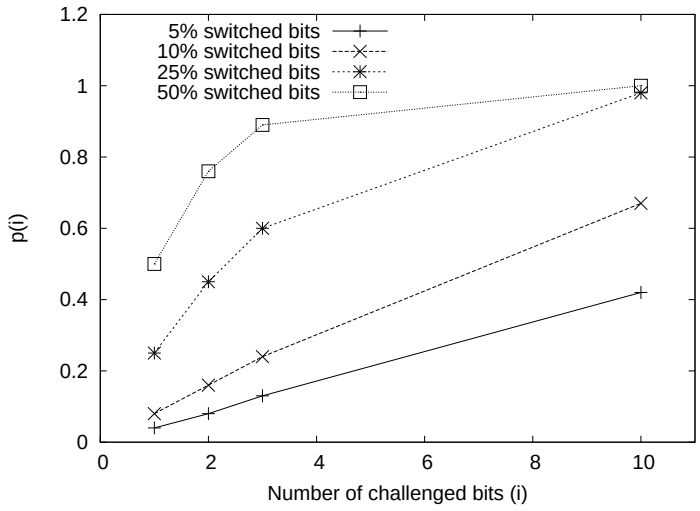

Figure 23: $p(x,i)$ when 5, 10, 25 and 50% of the 1-bits are switched. $p(x,i)$ increases as the lying percentage and the number of challenged bits increase.

However, the probability of detecting a lying peer does not only depend on $p(x,i)$ but also on how and when the challenges are sent by the peers. Our protocol does not explicitly specify when challenges should be sent to peers and how the system should react when a peer fails to reply to a challenge since these are strictly application-dependent. However, in this section, we outline a generic strategy for dealing with lying peers and based on this strategy analyze the probability and the time to detect them in a given system.

We assume that each peer is working with m peers on average for the needs of the application. A peer y can challenge a peer x as a step of one of their connections, and that a successful response is mandatory for any interaction afterwards. Hence, both peers should be awake (which is governed by the Markov chain depicted in Fig. 8). The probability that y is in ON state is $p_{ON}^{y} = \frac{\mu_y}{\lambda_y+\mu_y}$. Similarly, the probability that x is ON is p_{ON}^{x}. Since these probabilities are independent, the probability that x can be connected to and challenged by y, $p_c(x,y)$, is

$$p_c(x,y) = p_{ON}^{x} \cdot p_{ON}^{y} \tag{6}$$

Let's assume y challenges x with a known frequency, f. Depending on this frequency, the probability that the lying peer x is detected by y, $p_{detect}(x,y)$, during the system time t_s is:

$$p_{detect}(x,y) = 1 - (1 - p_c(x,y) \cdot p(x,i))^{f \cdot t_s} \tag{7}$$

In other words, a lying peer will only be *not* detected if all the successfully sent challenges have a successful response during t_s. Note that $p_{detect}(x,y)$ is simply the probability of finding the switched bits in the bit field of x. The actual detection happens when x cannot respond to the challenge (e.g., by not sending a proof or sending a false proof).

With m peers working with x and thus challenging x, the probability of detecting an lying peer x, $p_{detect}(x)$, is:

$$p_{detect}(x) = 1 - \Pi_{y=1}^{m}(1 - p_{detect}(x, y)) \tag{8}$$

Again, x will only be *not* detected, if all m peers fail to detect its lie.

In addition to detection probability, average detection time is also important as it affects how fast we can recover from availability measurement errors. To calculate the average detection time, t_{detect}, we first need to calculate the probability of detecting a lying peer x on the n^{th} try. We denote this probability as $p^n_{detect}(x, y)$ and calculate it as:

$$p^n_{detect}(x, y) = (1 - p_c(x, y) \cdot p(x, i))^{n-1} \cdot p_c(x, y) \cdot p(x, i) \tag{9}$$

Since this follows a geometric distribution, the mean number of tries necessary for y to detect x is $\frac{1}{p_c(x,y)\cdot p(x,i)}$. Since there are m peers challenging x, x will be detected whenever one of these peers discovers its lie. Hence, average detection time of x depends on the minimum of the average number of tries necessary among m peers. Since the time between each challenge is $\frac{1}{f}$, the average time to detect a lying peer, t_{detect} is:

$$t_{detect} = \frac{1}{f} \cdot \min_{y} \frac{1}{p_c(x, y) \cdot p(x, i)} \tag{10}$$

To understand if Pacemaker can detect lying peers efficiently, we study a network where each peer is challenged by $m = 5$ peers. Peers send a challenge once every day (i.e, $f = 1$) for a system time, $ts = 15$ days. Note that under uniform distribution, the average availability of a challenging peer is 51%, whereas this is 8% for exponential distribution. Furthermore, based on Fig. 23, i is selected as 3. We next analyze the $p_{detect}(x)$ when the lying peer x is 5%, 10%, 25% and 50% available in the system. Moreover, we assume x lies uniformly random based on its availability. In other words, if it is 5% available it tries to improve its availability by U(1%,95%). Based on this lying behaviour, the following (availability, average lying) values are analyzed: (0.05, 0.48), (0.1, 0.46), (0.25, 0.38) and (0.5, 0.26). Given this setting, using Eqs. 5-8, we plot $p_{detect}(x)$ in Fig. 24. As expected, due to the low average availability of peers, the probability of detecting a lying peer is lower for exponential distribution compared to uniform distribution. However, note that, for both cases, the probability of detection is high if the lying peers are online 50%. Actually, it is important to catch these peers since lying peers with less than 10% availability are not using the system anyway. Similarly, Fig. 25 shows that under uniform distribution, the lying peers are expected to be caught faster than exponential distribution. However, as a lying peer's presence increases in the system, the detection time decreases accordingly for both distributions.

Our analysis results are also confirmed by simulation results, which are depicted in Fig. 26 and Fig. 27 for uniform and exponential distributions, respectively. In our simulations, 5% of the population consists of lying peers. We wait for five days to reach a stable network before sending challenges (this is the reason why the number of lying peers stays flat in both figures upto 5 days and then starts decreasing). The results show that the lying peers are detected in accordance with the analysis: faster for uniform distribution and slower for

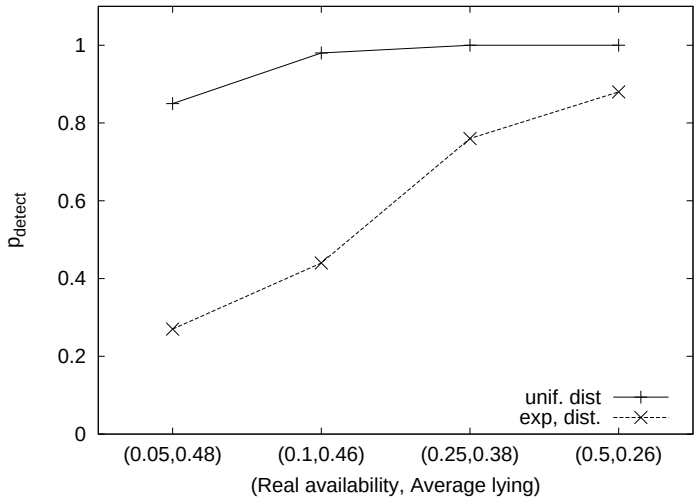

Figure 24: $p_{detect}(x)$ with different availability and lying characteristics

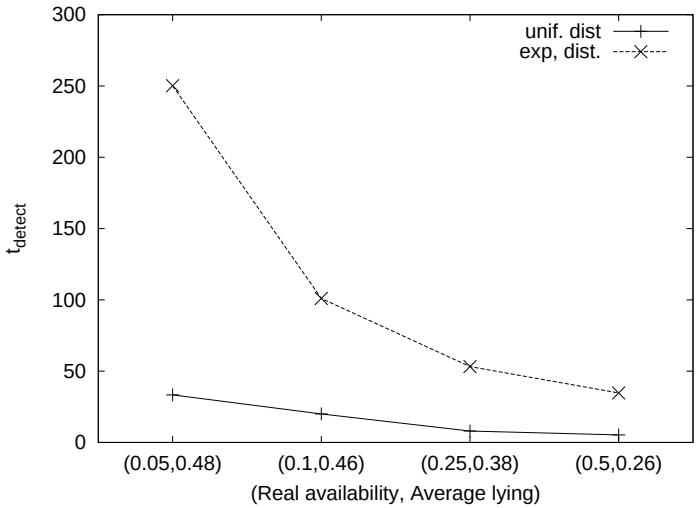

Figure 25: $t_{detect}(x)$ with different availability and lying characteristics

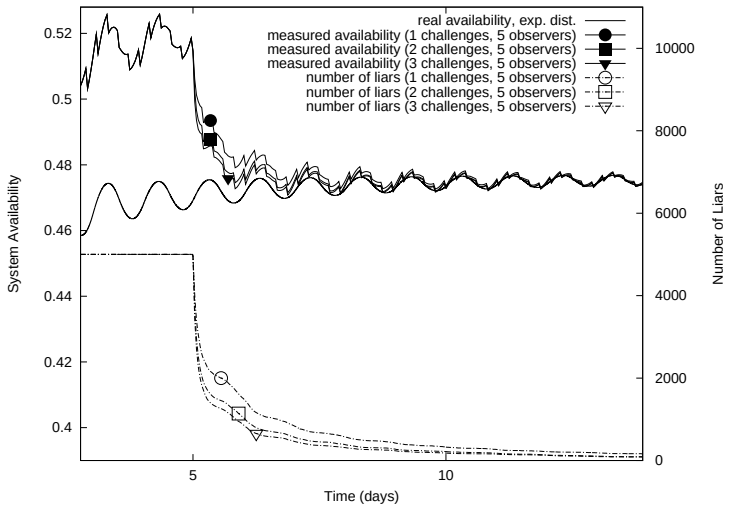

Figure 26: The measured and real availability, and the remaining number of lying peers in the system with time (for uniform distribution).

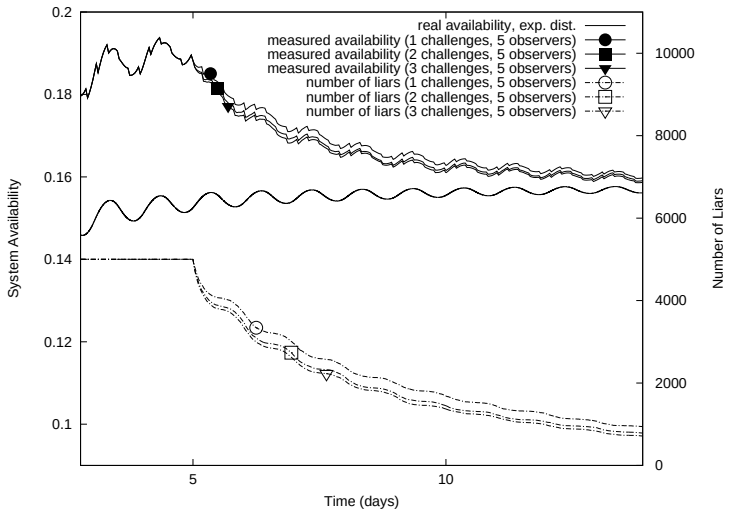

Figure 27: The measured and real availability, and the remaining number of lying peers in the system with time (for exponential distribution).

exponential distribution. More specifically, as predicted from Fig. 25, almost all lying peers are caught after the 10^{th} day (i.e., in 5 days) for uniform distribution. On the other hand, for exponential distribution, almost 20% liars are waiting to be detected after 15 days (see Fig. 27). However, we see that once the system starts detecting and removing lying peers from the network, the measured system availability approaches the real system availability for both distributions.

8 Implementation

As a first step towards real deployment, we ran Pacemaker over 170 nodes of the Planet-Lab network for one month since September 24, 2008. In this section, we present the details of our implementation and initial results.

8.1 Implementation Details

Pacemaker was implemented as a single program, written in Objective-Caml. It uses openSSL for cryptography and network libraries for peer-to-peer communications from our previous work (MLdonkey [14], Peerple [8]), specifically, for marshaling messages and establishing communications. One of the most important features of Pacemaker is its ease of deployability: It took a single programmer less than a week to implement and fully deploy it (including an additional auto-upgrade feature).

In our implementation, an option is used to decide the role of the node in the system at start-up. These roles are:

- **Client:** A standard peer in the network.
- **Server:** A *server* peer, which diffuses pulses in the network periodically.
- **Master:** A *logger* peer, which doesn't run the Pacemaker protocol but instead, all peers connect to it every hour to upload their logs. This is done to be able to analyze these logs to evaluate the performance of Pacemaker.

The entire system can be divided into three parts: (1) our Pacemaker protocol, (2) a mesh protocol for building the network and (3) a file sharing protocol for logging and software update purposes. All of these protocols contain 12 messages in total, as listed below:

- **Pacemaker:** 4 messages, which are `Pulse`, `Availability`, `Challenge` and `Proof`, and their handlers have been implemented. However, since we are not running any real application and have no selfish nodes, only `Pulse` messages are sent in our experiments.
- **Mesh:** The mesh protocol builds the underlying network using 3 messages. The `AskParent` and `AskParentReply` messages are used to establish permanent links between peers and propagate other parent candidates. The `Distance` message helps decreasing the diameter of the network.
- **File sharing:** Finally, we use 5 messages to transfer files between peers and synchronize directories. These messages allow:
  - Logging: The log directory of every peer is synchronized with the master. In each synchronization only new or modified files are transfered.
  - Software updates: To diffuse a new binary in the network, one of the clients is updated, which in turn starts a gossip of this update.

The final program is 2000 lines of Objective-Caml code, where 400 lines are message descriptions (among which 140 lines are for Pacemaker), 700 lines are handlers (among which only 70 lines are for Pacemaker; the file sharing feature is the most verbose) and 250 lines are for the main functionality.

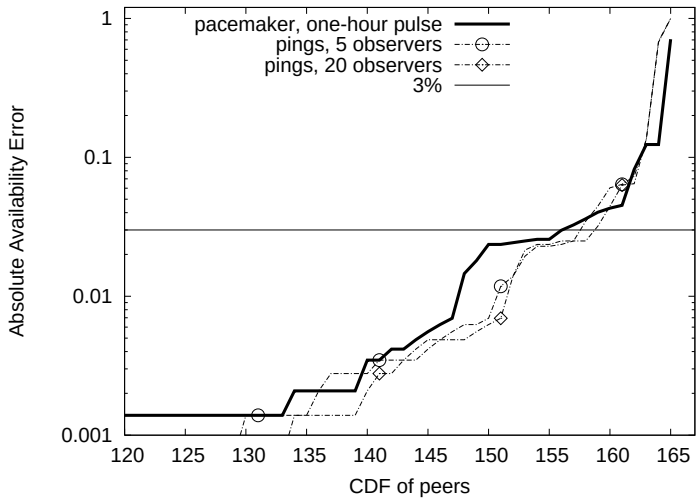

Figure 28: Availability Error in Planet-Lab evaluation, 165 nodes

8.2 Deployment Details

To deploy Pacemaker, we used three computers in our lab. One of these served as the *server* and also propagated software updates in the network. The second computer acted as a normal *client* to enable local debugging of problems. The third computer was the *master*. Next, we got access to one Planet-Lab slice, where we first started with 10 nodes, then 50 nodes after two days, and finally deployed Pacemaker on 170 nodes. However, due to a restarting problem with `crond` daemon after node crashes, the number of nodes running Pacemaker was observed to go down as low as 145 before the `crond` daemon is restarted manually. Therefore, the results presented in the next section are on one day including 165 nodes.

8.3 Results

The initial results with our Planet-Lab deployment are depicted in Fig. 28. These results show that using Pacemaker, the error in availability measure remained below 1% for 90% of the peers. Only for the 6% of the peers, the error was higher than 3%. We also deployed a ping-based availability measurement system to be able to compare it against Pacemaker. The figure shows that the ping-based system both with 5 and 20 observers perform similarly to Pacemaker in terms of accuracy. However, note that Pacemaker achieves this accuracy level with a lower cost. Furthermore, the similarity in accuracy performance is also not surprising because the availability of nodes in our slice did not show much variation. Comparing ping-based system with 5 observers against 20 observers also confirms this as the increase in the number of observers did not improve the availability measure. In the future, we plan to use Pacemaker in a real peer-to-peer backup storage system to evaluate its performance in more dynamic settings.

8.4 Discussion

Our goal with a Planet-Lab implementation was to show the ease-of-deployment of Pacemaker. However, since there were no selfish peers in the network, we were not able to test the availability notifications and the verification of peer availability. Nevertheless, it is important to evaluate the cost of Pacemaker when `Availability`, `Challenge` and `Proof` messages are sent. Especially the cost of bandwidth needs to be considered since it is usually the rarest resource in peer-to-peer networks.

In Pacemaker, bandwidth is mainly consumed during the exchange of signatures and keys in pulse messages. With RSA, these signatures and keys are typically 256 bytes long. This cost can further be reduced by using elliptic curves, which achieve a good level of security with around 15 bytes. Therefore, we do not expect Pacemaker to incur high costs. For instance, if we use RSA, the size of the messages sent by Pacemaker can be calculated approximately for a year (i.e, $N_t = 8760$) as:

- *Pulse* : 800 B (2 keys and 1 signature)
- *Availability* : 1400 B (bit field[N_t] + 1 signature)
- *Challenge* : 70 B (a nonce of 64 B)
- *Proof* : 900 B (nonce + 1 key + 2 signatures)

Note that the pulse message is sent once per period P to all the neighbors. Other messages would probably only be sent once a day between two peers working together. Consequently, we expect the bandwidth cost of Pacemaker to be negligible.

9 Related Work

In this section, we first present current research on peer-to-peer networks that relies on availability information. Such systems serve as our main motivation to provide availability information securely in the presence selfish peers. Next, we discuss related work on availability measurement, focusing specifically on their operation in the presence of selfish peers.

9.1 Uses of Availability Information in P2P Networks

The majority of the research on building peer-to-peer networks heavily relies on information about stability or availability of peers. Indeed, many systems rely on a *stable core* or *super-peers*, which are selected for their high availability in the system. For instance, [1] and [4] report that, in Gnutella and Overnet, respectively, the peers with higher availability tend to be more stable than other peers. Based on this result, [1] proposes a protocol that builds a more stable network by selecting peers with higher availability. However, the proposed solution cannot cope with selfish peers that might lie about their real availability to get a better status in the system. Similarly in [21], a gradient topology is built so that the most stable peers are at the *core* of the network and the less stable peers stay on the border. However, selfish peers can claim higher availability to be included in the *core*, and then refuse to serve requests even though they

	Pacemaker	AVMON
Architecture	Pulses	Pings
Durability	Unlimited	Limited (churn)
Supported Topologies		
Internet	Yes	Yes
Ad-hoc Networks	Yes	No
Social Networks	Yes	No
Firewalled Peers	Yes	No
Vulnerabilities		
Selfish Peers	Resilient	Not Treated
Colluding Peers	Not Treated	Vulnerable

Figure 29: Comparison with AVMON [19]. See section 9.2 for details.

benefit from their good position in the network. In [9], availability information is used to select *super-peers* in the network to build a top-level Chord Distributed Hash Table (DHT) over another less stable DHT. Again, selfish peers might manage to be included in the top-level DHT, and use their position to decrease their load.

In addition to help build more stable networks, availability information is also useful for repairing the network. For instance, using availability information, replacement policies decrease the effect of churn in a peer-to-peer system [10]. It was shown that, although performing well, the performance of a random replacement cannot reach the performance of a replacement policy based on choosing peers with maximum availability. Similarly, in [7], availability information is used to proactively repair fragments in a peer-to-peer storage system based on an estimation of the failure rate. Finally, in [22] an object replica maintenance system is studied under temporary and permanent failures for different peer-to-peer systems. It is shown that data tends to accumulate on nodes with high availability and unlimited capacity. Essentially, if the capacity is limited, the performance degrades when the nodes with high availability become saturated as the nodes with low availability trigger many repairs. Obviously, if the availability information is compromised in any of these systems, repairs would not be possible.

9.2 Comparison with AVMON

Availability measurement systems can be classified into two categories: *clock-based* systems, where measures are based on the local clock and *ping-based* systems where measures are done by hello messages. Pacemaker introduces a new category, *pulse-based* systems, where measures are based on pulses flooded in the network. Clock-based systems such as [18] are obviously vulnerable to selfish peers.

We introduced and evaluated ping-based systems in Section 6.3. Our results show that, to get the same level of accuracy as Pacemaker in a realistic system, a peer in a ping-based system needs to send 25 ping messages per hour, while a peer in Pacemaker only needs to send 5 diffusion messages per hour. Indeed, in ping-based systems, peers can only monitor availability when they are online, so more monitors are needed to cope with churn. Moreover, their

measures become unavailable as soon as they leave the system. Finally, peers behind firewalls cannot be monitored, whereas Pacemaker can still reach them. Therefore, measurements are less accurate, less efficient and less durable.

To the best of our knowledge, AVMON [19] is the only ping-based system designed with security in mind. By using a hash function to match observers and observed peers, AVMON tries to avoid that peers claim higher availability than the reality by colluding with other peers. AVMON suffers from both the drawbacks of ping-based systems and the drawbacks of its hash-based scheme, as detailed in [13]. From a security point-of-view, selfish observers can still lie about the availability of the peers that they are supposed to monitor. Moreover, the hash mechanism is vulnerable to collusion: peers can change their listening port until they are accepted as observers for the peers with whom they want to collude.

9.3 Colluding Peers

As other availability measurement systems, Pacemaker cannot yet cope with colluding peers, but it is, to the best of our knowledge, the only one resilient to selfish peers. Nevertheless, we think that, first, collusion is harder to implement than selfishness, and second, it should be possible to extend Pacemaker to cope with collusion.

Indeed, selfishness requires only small modifications of the software (to lie on bitmaps) or of the environment (to filter out pings or requests in ping-based systems). On the contray, collusion requires deep modifications of the software (extension of the protocol) and complicity of other peers that need to be discovered on the network. For the first part, there is a game-theoric incentive not to diffuse such modified software for colluding: the bigger the number of peers colluding, the smaller the benefit for each colluding client. Moreover, clients trying to discover other colluding clients openly can be detected by honey-pots, i.e. clients accepting to collude only to detect colluders, and able to blacklist them on a system-wide scale. Consequently, collusion can be expected to be limited to a few manually created groups of modified clients trusting each other.

There are also several approaches to extend Pacemaker to cope with collusion. A first approach would be to insert in the pulse the path of IP addresses followed during its diffusion. Such a scheme would help blacklisting clients that keep diffusing old pulses to other peers to disrupt the system. Another approach would be to track modifications of the history of pulses of a client, to detect if an old pulse is added, to limit the time during which collusion can happen. Finally, we are also investigating a more interesting approach, closer to Pacemaker spirit, based on the use of Merkle trees [17], to actually encode the presence of a peer in the system directly in the pulse.

10 Conclusion

In this paper, we have presented a simple but efficient way of monitoring availability in peer-to-peer systems in the presence of selfish peers. Our protocol, Pacemaker, uses a set of servers to propagate cryptographic pulse messages in a mesh of peers, allowing them to measure and check the history of availability of

other peers easily at any time. Pacemaker is resistant to selfish behaviors, and in particular to lying peers, which lie about their availability to gain access to more resources in the system.

We have evaluated Pacemaker through analysis, simulations and deployed the protocol on a Planet-Lab testbed. Our results show that Pacemaker is accurate, able to detect selfish behaviors, less expensive than competitors and easy to deploy. Furthermore, the low overhead induced by Pacemaker enables not to hamper the scalability of the peer-to-peer overlay network.

Pacemaker also introduces a new network architecture, *pulse-based* systems, that, we think, could have multiple applications in self-organizing systems. We are now investigating some of these applications, for example in the context of sensor networks. As discussed in section 9.3, we are also working on different approaches to extend Pacemaker to cope with colluding peers.

References

[1] Ranjita Bhagwan, Stefan Savage, and Geoffrey Voelker. Understanding availability. In *IPTPS, Int'l Work. on Peer-to-Peer Systems*, 2003.

[2] Ranjita Bhagwan, Kiran Tati, Yu-Chung Cheng, Stefan Savage, and Geoffrey M. Voelker. Total recall: system support for automated availability management. In *NSDI, Symp. on Networked Systems Design and Implementation*, 2004.

[3] William J. Bolosky, John R. Douceur, David Ely, and Marvin Theimer. Feasibility of a serverless distributed file system deployed on an existing set of desktop pcs. In *SIGMETRICS, Int'l Conf. on Measurement and Modeling of Computer Systems*, 2000.

[4] Fabian E. Bustamante and Yi Qiao. Friendships that last: peer lifespan and its role in p2p protocols. In *Int'l workshop on Web content caching and distribution*, 2004.

[5] Frank Dabek, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Wide-area cooperative storage with CFS. In *SOSP, ACM Symposium on Operating Systems Principles*, 2001.

[6] T. Dierks and E. Rescorla. RFC 4346: The Transport Layer Security (TLS) protocol version 1.1. IETF, April 2006.

[7] Alessandro Duminuco, Ernst W Biersack, and Taoufik En Najjary. Proactive replication in distributed storage systems using machine availability estimation. In *CoNEXT, Int'l Conf. on emerging Networking EXperiments and Technologies*, 2007.

[8] Anh-Tuan Gai, Fabrice Le Fessant, and Laurent Viennot. http://www.peerple.net/, 2007.

[9] L. Garcias-Erice, E.W. Biersack, P.A. Felber, K.W. Ross, and G. Urvoy-Keller. Hierarchical peer-to-peer systems. In *Euro-Par, Intl. Conf. on Parallel and Distributed Computing*, 2003.

[10] P. Brighten Godfrey, Scott Shenker, and Ion Stoica. Minimizing churn in distributed systems. In *SIGCOMM, Int'l Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communications*, 2006.

[11] Saikat Guha, Neil Daswani, and Ravi Jain. An Experimental Study of the Skype Peer-to-Peer VoIP System. In *IPTPS, Int'l Work. on Peer-to-Peer Systems*, 2006.

[12] C. Hendrik. Routing information protocol. `http://www.ietf.org/rfc/rfc1058.txt`, June 1998.

[13] Fabrice Le Fessant. Limitations of the AVMON system. http://fabrice.lefessant.net/AVMON/.

[14] Fabrice Le Fessant. http://mldonkey.sourceforge.net/, 2002.

[15] Arjen K. Lenstra and Eric R. Verheul. Selecting cryptographic key sizes. In *PKC, Int'l Work. on Practice and Theory in Public Key Cryptography*, 2000.

[16] Petar Maymounkov and David Mazières. Kademlia: A peer-to-peer information system based on the XOR metric. In *IPTPS, Int'l Work. on Peer-to-Peer Systems*, 2002.

[17] Ralph Merkle. *Secrecy, authentication, and public key systems.* PhD thesis, 1979.

[18] James W. Mickens and Brian D. Noble. Exploiting availability prediction in distributed systems. In *NSDI, Symp. on Networked Systems Design and Implementation*, 2006.

[19] Ramses Morales and Indranil Gupta. AVMON: Optimal and scalable discovery of consistent availability monitoring overlays for distributed systems. In *ICDCS: Int'l Conf. on Distributed Computing Systems*, 2007.

[20] Bogdan C. Popescu, Bruno Crispo, and Andrew S. Tanenbaum. Safe and private data sharing with turtle: Friends team-up and beat the system. In *SPW, Security Protocols Work.*, 2004.

[21] Jan Sacha, Jim Dowling, Raymond Cunningham, and René Meier. Discovery of stable peers in a self-organising peer-to-peer gradient topology. In *DAIS, IFIP Int'l Conf. Distributed Applications and Interoperable Systems*, 2006.

[22] Kiran Tati and Geoffrey M. Voelker. On object maintenance in peer-to-peer systems. In *IPTPS, Int'l Work. on Peer-to-Peer Systems*, 2006.

[23] Jing Tian and Yafei Dai. Understanding the dynamic of peer-to-peer systems. In *IPTPS, Int'l Work. on Peer-to-Peer Systems*, 2007.

[24] Spyros Voulgaris, Daniela Gavidia, and Maarten van Steen. CYCLON: Inexpensive membership management for unstructured P2P overlays. *J. Network System Management.*, 13(2), 2005.

Contents

1 Introduction **3**

2 Pacemaker in a nutshell **4**

3 Model **5**
3.1 Definitions 5
3.2 Network Model 5
3.3 Cryptographic Model 6

4 Pacemaker in Detail **7**
4.1 Pulse dissemination protocol 7
4.2 Inquiry Protocol 8
4.3 Verification Protocol 9

5 Evaluation Road-map **10**

6 Availability Monitoring with Pacemaker **11**
6.1 Simulating the Availability of Peers 11
6.2 Overlay Network Setup 13
6.3 Results on Accuracy 16

7 Pacemaker against Selfish Peers **19**
7.1 Lazy Peers 20
7.2 Opportunistic Peers 21
7.3 Lying Peers 22

8 Implementation **27**
8.1 Implementation Details 27
8.2 Deployment Details 28
8.3 Results 28
8.4 Discussion 29

9 Related Work **29**
9.1 Uses of Availability Information in P2P Networks 29
9.2 Comparison with AVMON 30
9.3 Colluding Peers 31

10 Conclusion **31**

Centre de recherche INRIA Saclay – Île-de-France
Parc Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 Orsay Cedex (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399